ATI-000118BN

Express Mail Label No. EL566348792

VERTEX DATA PROCESSING WITH MULTIPLE THREADS OF EXECUTION

BACKGROUND

The addressable and displayable basic element used to build up a computer image is a pixel. Each pixel has several essential parameters stored as the pixel's vertex data. Typical parameters are position data, such as an X coordinate, a Y coordinate and a Z coordinate, that indicate the pixel's reference position in three dimensions (3D); color information, such as diffuse color parameters (R_D, G_D, B_D, A) and specular color parameters (R_S, G_S, B_S, F) which form the pixel's diffuse color and specular color; texture information, such as the pixel's texture pattern and the depth of the pattern from the viewer; or any other suitable information needed by the specific individual application. Based on the graphic standards used by an application, parameters may be stored in different orders or formats within the vertex data. For example, coordinate parameters may be stored as 32-bit floating-point format or fixed-point format. The color information parameters may be stored as a simple group of 4 bytes or as a complicated group of 16 bytes. The graphic device displays the pixel based on its vertex data parameters.

Typical image display systems by using hardware and software have automated several primitive draw functions. For example as shown in Figure 1a, to draw a line, the application needs to provide only the beginning pixel point A 10 (X_1, Y_1, Z_1) and the ending pixel point B 12 (X_2, Y_2, Z_2) to the graphic device 9. The graphic device 9 determines which pixels are on the line between pixel A 10 and pixel B 12. Subsequently, the graphic device 9 sets up these pixels' color information using the A and B pixels' color parameters. If the

application wants to move the line to a new location, the new positions of A 10 will be A′ 14 (X_1+a, Y+b, Z_1) and B 12 will be B′ 16(X_2+a, Y_2+b, Z_2). If a scaling factor c is involved, the new A′ 14 pixel will be (x_1*c+a, y_2*c+b, z_2) and B′ 16 will be (X_2*c+a, Y_2*c+b, Z_2).

The same principle applies to drawing a triangle, another primitive function. An application provides vertex data that has parameters of the three triangle end points. The graphic device 9 will set up the vertex data of all relevant pixels to draw the triangle. All two dimensional (2D) or 3D graphic objects are made up of a number of polygons which can be broken into primitive functions, such as lines, triangles etc. To redraw 2D or 3D graphic objects requires redrawing the relevant primitives. The redrawing requires setting up all corresponding pixels' vertex data and redrawing them. All graphic operations, simple or complicated, are performed by manipulating the contents of pixel vertex data by multiplication, addition or logical operations, such as OR and exclusive OR.

Users of personal computers or game systems utilize real-time effects on displayed images. In such systems, a 2D or 3D image is displayed at a rate of 30 or more frames per second. These rates allow the user to perceive continuous motion of objects in a scene. To achieve such a real-time, realistic and interactive image requires a tremendous amount of processing power. These effects require processing over a million graphic primitives per second. Typically, processing a million primitives requires multiplying and adding millions of floating-point and fixed-point values.

Accordingly, it is desirable to improve the efficiency of transforming vertex data.

SUMMARY

Multi-thread video data processing for use in a computer video display system. The parameters of vertex data are grouped into a plurality of groups. The computation needs of each group are broken down into several arithmetic operations to be performed by corresponding arithmetic units. The units concurrently process the vertex data.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1a illustrates two displayed line images.

Figure 1b is the vertex data of the lines of Figure 1a.

Figure 2 illustrates functional blocks of a setup engine.

Figure 3a is a table of the basic state operations for the position data group.

Figure 3b is a state diagram for the position data group.

Figure 4a is a table of the basic state operations for the color information group.

Figure 4b is the state diagram for the color information group.

Figure 5a is a table of the basic state operations for the texture information group.

Figure 5b is the state diagram flow chart for the texture information group.

Figure 6 illustrates the functional process flow for the transform engine.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Instead of using a traditional sequential processing approach, a multi-thread approach to process the vertex data may be used. As shown in Figures 1a and 1b, computer monitor 9 displays a first line with the beginning pixel point A 10 with parameters X_0, Y_0, Z_0, W_0, S_0, T_0, C_0 and the end pixel point B 12 with parameters X_1, Y_1, Z_1, W_1, S_1, T_1 and C_1 stored as vertex data 20. That line may be modified. It may be moved to a new location, such as to

begin point 14 and end point 16. It may be scaled. It may have its specular color and texture pattern modified. One approach to redrawing the line is to process all parameters of vertex data 20 into new vertex data 40 before the new vertex data 40 is submitted for the line redraw.

The transform process will be explained with reference to modifying a line's pixel vertex data parameters. This transform process may be used for any transformation. As shown in Figure 2, the transform engine 67 is a part of a setup engine 65. Vertex data is transformed by the transform engine 67 and processed by the other data processing engine 68. Subsequently, the transformed and processed data is sent to raster engine 69 prior to output to the monitor 9.

The transform engine 67 initially groups vertex data parameters together for processing. The groups allow for more efficient utilization of each arithmetic unit, such as a floating-point multiplication unit and a floating-point addition unit. One grouping scheme groups: the pixel position vertex data, the pixel color vertex data and the pixel texture vertex data together. To illustrate for a line, the pixels' position data X_0, Y_0, Z_0 and W_0 and X_1, Y_1, Z_1 and W_1 is selected as a first group. The pixels' color data C_0 and C_1 is selected as a second group and the pixels' texture data S_0, T_0 and S_1, T_1 is selected as a third group. By analyzing the computational requirements of each group, the required tasks can be broken down into addition and multiplication operations. The broken down operations are used to construct multiplication and addition state operations. Any computation needs of the group can be fulfilled by using the combination of its basic state operations to achieve the final results. Using sequential states, the addition unit may perform operations such as subtraction, move, floating-point number conversion to fixed number, truncate, round to even, round to odd.

To transform the position data group as shown in Figure 3a, one approach is to use ten basic state operations 80-89. Six 80-85 out of the ten basic 80-89 state operations involve multiplication. Three state operations 86-88 involve addition and one state operation 89 is a wait, no operation (NOP), state operation. There is also an idle state 79. As shown in Figure 3a, position state operation 0 80 involves multiplying the X coordinate by a scale factor. Position state operation 8 88 involves adding the Z coordinate with an offset. For vertex data of the initial line begin pixel A 10 (X_0,Y_0, and Z_0) transforms to A_1 14 ($X_0^{/}$ $=X_{0*}c_1+a_1$, $Y_0^{/}=Y_{0*}c_2+a_2$, $Z_0^{/}=Z_{0*}c_3+a_3$). The transformation will require position state operations (PSO) 0, 6, 1, 7, 2 and 8; 80, 86, 81, 87, 82 and 88 to complete the whole computation. Referring back to Figure 3b, the different paths from one position state operation to other position data state operations are shown.

To transform the color data group, one approach is to use ten independent color state operations (CSO), as shown in Figure 4a. Each CSO involves only addition with one color parameter. CSO 0-3 100-102 are related to diffuse color parameters addition, CSO 4-7 104-107are related to specular color parameters addition, and CSO 8-9 108-109 move the R_s and R_d vertex data. The move operation may be performed using an addition unit. The different paths from one color state operation to other color state operations are shown in Figure 4b. To transform the texture data group, one approach is to use eight texture state operations (TSOs). Six 122-127 of the TSOs are multiplication related and two 120, 121 of the TSOs are moves which can be performed by addition. Figure 5a shows the different paths from one TSO 120-127 to other TSOs 120-127.

By grouping the vertex data into position, color and textural groups, multiple arithmetic units, such as a floating-point multiplication and a floating-point addition unit, may be utilized more efficiently. To illustrate, if position group data is utilizing the floating-multiplication unit to perform a multiplication operation, simultaneously an addition operation of either the color group or texture group can utilize the addition unit. By continuously sending multiplication and addition operations to queues associated with the multiplication and addition units, both the multiplication and addition unit are used with higher efficiency accelerating data processing.

Each of these groups of operations comprise a "program", or "thread of execution" that vies for the use of the shared arithmetic resources. Multiple controllers are typically used, each executing a thread, that can generate a sequence of instruction for the shared arithmetic resources.

It is a common requirement that the vertex data processor be flexible enough, via programmability, to perform a certain subset of all of its possible operations, for any given graphics primitive or vertex. Since the exact operations to be performed by the transform engine are not known until run-time, it is desirable for the processor to respond dynamically to the processing workload to efficiently use the available processing resources. One technique for dynamic processing is to group the operations based on which function unit they use. Subsequently, the operations are concurrently scheduled to each function unit.

To illustrate as shown in Figure 6, the vertex data 140 is broken into three groups; position group 145, color group 150 and texture group 155. The position group 145 requires PSO 0, 6, 1, 7, 2 and 8; 80, 86, 81, 87, 82 and 88 to complete its data transformation. The color group 150 requires CSO 0 and 8; 100 and 108 to complete its transformation. The textural group 150 requires TSO 0 and 2; 120 and 122 to transform the textural parameters.

All multiplication state operations from the position or textural groups 145, 155 will be queued at the multiplication queue 160 and all addition state operations from all three groups 145, 150, 155 will be queued at the addition queue 165. The queued operations of both queues 160, 165 will be independently executed by the multiplier unit 170 and the adder unit 175. The queues are controlled by schedulers, such as an M-scheduler 181 and A-scheduler 182.

In certain circumstances, coordination between threads is needed. For example, intermediate results from the position thread (for example, perspective-related information) may be required by the texture thread. Binary or counting semaphore 180 can be used to synchronize the sequential execution of two different threads and to signal when the result from one thread is available for the next thread to consume. The results of the executed operations are sent to a post-processing engine 185, such as the XEOPIPE, which performs operations, such as rounding or conversion from floating-point to fixed-point format. The buffer 195 holds the transformed vertex data until required by other processes.

* * *